12th IFAC/IEEE Workshop on Discrete Event Systems
Cachan, France. May 14-16, 2014

A Symbolic Approach for Maximally Permissive Deadlock Avoidance in Complex Resource Allocation Systems

Zhennan Fei * **Spyros Reveliotis** ** **Knut Åkesson** *

* *Chalmers University of Technology, Gothenburg, Sweden (e-mail: {zhennan, knut.akesson}@chalmers.se)*
** *Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: spyros@isye.gatech.edu)*

Abstract: To develop an efficient implementation of the maximally permissive deadlock avoidance policy (DAP) for complex resource allocation systems (RAS), a recent approach focuses on the identification of a set of critical states of the underlying RAS state-space, referred to as minimal boundary unsafe states. The availability of this information enables an expedient one-step-lookahead scheme that prevents the RAS from reaching outside its safe region. This paper presents a symbolic approach that provides those critical states. Furthermore, by taking advantage of certain structural properties regarding RAS safety, the presented method avoids the complete exploration of the underlying RAS state-space. Numerical experimentation demonstrates the efficiency of the approach for developing the maximally permissive DAP for complex RAS with large structure and state-spaces, and its potential advantage over similar approaches that employ more conventional representational and computational methods.

Keywords: Resource Allocation System, Discrete Event System, Deadlock Avoidance, Maximal Permissiveness, Supervisory Control Theory, Binary Decision Diagram.

1. INTRODUCTION

In the Discrete Event Systems (DES) literature, the concept of the Resource Allocation System is a well-established abstraction for modeling the resource allocation dynamics that take place in many technological applications (Reveliotis (2005); Zhou and Fanti (2004); Campos et al. (2014)). Following those past developments, in this work, we define a *resource allocation system (RAS)* [1] by a 4-tuple $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$ where: (i) $\mathcal{R} = \{R_1, \ldots, R_m\}$ is the set of the system *resource types.* (ii) $C : \mathcal{R} \rightarrow \mathbb{Z}^+$ – where $\mathbb{Z}^+$ is the set of strictly positive integers – is the system *capacity* function, characterizing the number of identical units from each resource type available in the system. Resources are assumed to be *reusable*, i.e., each allocation cycle does not affect their functional status or subsequent availability, and therefore, $C(R_i) \equiv C_i$ constitutes a system *invariant* for each R_i. (iii) $\mathcal{P} = \{J_1, \ldots, J_n\}$ denotes the set of the system *process types* supported by the considered system configuration. Each process type J_j, for $j = 1, \ldots, n$, is a composite element itself; in particular, $J_j = \langle \mathcal{S}_j, \mathcal{G}_j \rangle$, where $\mathcal{S}_j = \{\Xi_{j1}, \ldots, \Xi_{j,l(j)}\}$ denotes the set of *processing stages* involved in the definition of process type J_j, and $\mathcal{G}_j$ is an *acyclic digraph* that defines the sequential logic of process type J_j. The node set of $\mathcal{G}_j$ is in one-to-one correspondence with the processing-stage set $\mathcal{S}_j$, and each directed path from a source node to a terminal node of $\mathcal{G}_j$ corresponds to a possible execution sequence (or "process plan") for process type J_j. Also, given an edge $e \in \mathcal{G}_j$ linking Ξ_{jk} to $\Xi_{jk'}$, we define $e.src \equiv \Xi_{jk}$ and $e.dst \equiv \Xi_{jk'}$, i.e., $e.src$ and $e.dst$ denote respectively the source and the destination nodes of edge e. (iv) $\mathcal{A} : \bigcup_{j=1}^{n} \mathcal{S}_j \rightarrow \prod_{i=1}^{m} \{0, \ldots, C_i\}$ is the *resource allocation function*, which associates every processing stage Ξ_{jk} with the *resource allocation request* $\mathcal{A}(j,k) \equiv \mathcal{A}_{jk}$. More specifically, each $\mathcal{A}(j,k)$ is an m-dimensional vector, with its i-the component indicating the number of resource units of resource type R_i necessary to support the execution of stage Ξ_{jk}. Furthermore, it is assumed that $\mathcal{A}_{jk} \neq \mathbf{0}$, i.e., every processing stage requires at least one resource unit for its execution. Finally, according to the applying resource allocation protocol, a process instance executing a processing stage Ξ_{jk} will be able to advance to a successor processing stage $\Xi_{jk'}$, only after it is allocated the resource differential $(\mathcal{A}_{jk'} - \mathcal{A}_{jk})^+$; and it is only upon this advancement that the process will release the resource units $|(\mathcal{A}_{jk'} - \mathcal{A}_{jk})^-|$, that are not needed anymore. [2]

The "hold-while-waiting" protocol that is described above, when combined with the arbitrary nature of the process routes and the resource allocation requests that are supported by the considered RAS model, can give rise to resource allocation states where a set of processes are waiting upon each other for the release of resources that are necessary for their advancement to their next processing stage. Such persisting cyclical-waiting patterns are known as *(partial) deadlocks* in the relevant literature, and to the extent that they disrupt the smooth operation of the underlying system, they must be recognized and eliminated from the system behavior. The relevant control problem is known as *deadlock avoidance*, and a natural framework for its investigation is that of DES Supervisory Control Theory (SCT) (Ramadge and Wonham (1989), Cassandras and Lafortune (2008)). More specifically, in an Finite State

[1] The considered RAS class is known as the class of Disjunctive/Conjunctive RAS in the relevant literature, since it enables routing flexibility for its process types and requests for arbitrary resource sets at the various processing stages.

[2] We remind the reader that $x^+ = \max\{0, x\}$ and $x^- = \min\{0, x\}$.

Automaton (FSA) representation of the RAS dynamics, deadlocks appear as states containing a set of activated process instances and no feasible process-advancing events. Hence, assuming that the desired outcome of any run of this FSA is the access of the state where all processes have successfully completed and the underlying RAS is idle and empty of any active processes, the presence of deadlock states can be perceived as *blocking* behavior. Therefore, in the context of SCT, effective deadlock avoidance translates to the development of the *maximally permissive non-blocking supervisor* for the RAS-modeling FSA, that will confine the RAS behavior in the "trim" of this FSA, i.e., to the subspace consisting of the states that are reachable and co-reachable to the RAS idle and empty state.

In the relevant RAS theory, states that are co-reachable to the RAS idle and empty state are also characterized as *safe*, and, correspondingly, states that are not co-reachable are characterized as *unsafe*. Of particular interest in the implementation of the maximally permissive non-blocking supervisor for the considered RAS are those transitions leading from safe to unsafe states, since their effective recognition and blockage can prevent entrance into the unsafe region. The unsafe states that result from such problematic transitions are known as the *boundary unsafe* states in the relevant literature. Furthermore, for reasons that will be explained in the sequel, the entire set of the boundary unsafe states can be effectively recognized from its minimal elements. Hence, a particular approach for the implementation of the maximally permissive deadlock avoidance policy (DAP) for any instantiation of the aforementioned RAS class reduces to the effective enumeration and the efficient storage of the minimal boundary unsafe states of the underlying state space. This approach has been extensively investigated in the recent years, and the major results together with the supporting literature are systematically discussed in Reveliotis and Nazeem (2013).

The work presented in this paper seeks to complement the aforementioned past developments by introducing symbolic methods for the representation of the involved dynamics and of the target sets, and for the execution of the necessary computation. It is well known that, when properly balanced, symbolic representations of a DES state space can effect a dramatic compression of the information that is expressed by this state space compared to its more conventional representations. Furthermore, this compression can also lead to a significant speed-up of the computational algorithms that process this information for various analysis and (control) synthesis purposes. Indeed, the computational results that are presented at the end of this manuscript corroborate these expectations, and demonstrate clearly the effected gains in terms of computational time and memory requirements. On the other hand, due to the imposed page limits, the rest of this document is a rather minimal exposition of the pursued approach and the obtained results. A more expansive and thorough treatment of this material can be found in Fei et al. (2013).

2. PRELIMINARIES

2.1 Extended Finite Automata

The presented work employs the extended finite automaton (EFA) (Sköldstam et al. (2007)) as a formal representation of the RAS dynamics. An EFA is an augmentation of the ordinary FSA model with integer variables that are employed in a set of guards and are maintained by a set of actions. A transition in an EFA is enabled if and only if its corresponding guard is true. Once a transition is taken, updating actions on the set of variables may follow. By utilizing these two mechanisms, an EFA can represent the modeled behavior in a conciser manner than the ordinary FSA model.

More formally, an *Extended Finite Automaton (EFA)* over a set of model variables $v = (v_1, \ldots, v_n)$ is a 5-tuple $E = \langle Q, \Sigma, \rightarrow, s_0, Q^m \rangle$ where (i) $Q : L \times \mathcal{D}$ is the extended finite set of states. L is the finite set of the model *locations* and $\mathcal{D} = \mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ is the finite domain of the model *variables* $v = (v_1, \ldots, v_n)$. (ii) Σ is a nonempty finite set of events (also known as the alphabet of the model). (iii) $\rightarrow \subseteq L \times \Sigma \times G \times A \times L$ is the transition relation, describing a set of transitions that take place among the model locations upon the occurrence of certain events. However, these transitions are further qualified by G, which is a set of guard predicates defined on $\mathcal{D}$, and by A, which is a collection of actions that update the model variables as a consequence of an occurring transition. Each action $a \in A$ is an n-tuple of functions $(a_1, \ldots, a_n)$, with each function a_i updating the corresponding variable v_i. (iv) $s_0 = (\ell_0, v_0) \in L \times \mathcal{D}$ is the initial state, where ℓ_0 is the initial location, while v_0 denotes the vector of the initial values for the model variables. (v) $Q^m \subseteq L^m \times \mathcal{D}^m \subseteq Q$ is the set of marked states. $L^m \subseteq L$ is the set of the marked locations and $\mathcal{D}^m \subseteq \mathcal{D}$ denotes the set of the vectors of marked values for the model variables. For the sake of brevity, in the following, we shall use the notation $\ell \overset{\sigma}{\rightarrow}_{g/a} \ell'$ as an abbreviation for $(\ell, \sigma, g, a, \ell') \in \rightarrow$.

EFA-based modeling of RAS dynamics The formal construction of an EFA $E(\Phi)$ modeling the dynamics of any given RAS instance $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$ is presented in Fei et al. (2011). For the needs of this manuscript, this construction is briefly illustrated in the following example. The RAS instance Φ under consideration is shown in Fig.1. It comprises two process types J_1 and J_2, each of which is defined as a sequence of three processing stages; the stages of process type J_j, $j = 1, 2$, are denoted by Ξ_{jk}, $k = 1, 2, 3$. The system resource set is $\mathcal{R} = \{R_1, R_2, R_3\}$, with capacity $C_i = 1$ for $i = 1, 2, 3$. Each processing stage Ξ_{jk} requests only one unit from a single resource type; the relevant resources are depicted in Fig.1.

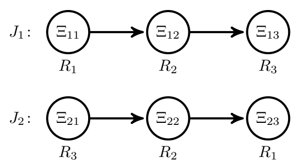

Fig. 1. A simple RAS

In the approach of Fei et al. (2011), each process type is modeled as a distinct EFA. Fig. 2 shows the EFA that models the behavior of process J_1 in the RAS of Fig. 1. This EFA has only one location, and its three transitions correspond to the loading and the process-advancing events among its different stages. On the other hand, since a process instance that has reached its final stage can always leave the system without any further resource requests, the unloading event is modeled only implicitly through the event that models the process access to its terminal stage(s). More specifically, in the EFA depicted in Fig. 2, the evolution of a process instance through the various processing stages is traced by the *instance variables* v_{1j}, $j = 1, 2$; each of these variables counts the number of process instances that are executing the corresponding processing stage. The model does not avail of a variable v_{13} since it is assumed that a process instance reaching stage Ξ_{13} is (eventually) unloaded from the system, without the need for any further resource allocation action.

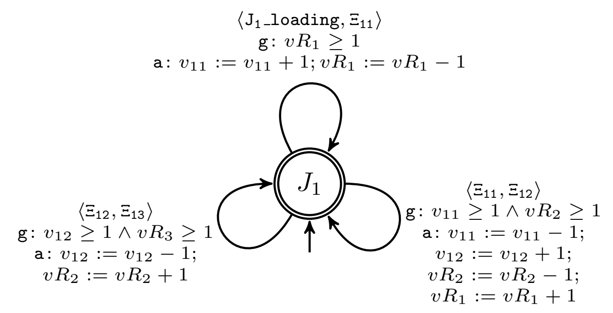
$\langle \mathtt{J_1\_loading}, \Xi_{11}\rangle$
$\mathtt{g}\colon vR_1 \geq 1$
$\mathtt{a}\colon v_{11} := v_{11}+1; vR_1 := vR_1 - 1$
J_1
$\langle \Xi_{12}, \Xi_{13}\rangle$
$\mathtt{g}\colon v_{12} \geq 1 \wedge vR_3 \geq 1$
$\mathtt{a}\colon v_{12} := v_{12} - 1;$
$vR_2 := vR_2 + 1$
$\langle \Xi_{11}, \Xi_{12}\rangle$
$\mathtt{g}\colon v_{11} \geq 1 \wedge vR_2 \geq 1$
$\mathtt{a}\colon v_{11} := v_{11} - 1;$
$v_{12} := v_{12} + 1;$
$vR_2 := vR_2 - 1;$
$vR_1 := vR_1 + 1$

Fig. 2. The resource-augmented EFA for J_1

The aforementioned EFA $E(\Phi)$ that model the process types J_1 and J_2 are linked through the global *resource variables* vR_i, $i = 1, 2, 3$, where each variable vR_i denotes the number of free units of resource R_i. Obviously, the domain of variable vR_i is $\{0, \ldots, C_i\}$.

Since, under proper RAS operation, the initial and the target final state correspond to the empty state, both the initial and the marked values of each variable vR_i are equal to C_i, and the corresponding values for all instance variables v_{jk} are equal to zero.

Finally, as depicted in Fig. 2, the resource and the instance variables are used to construct the necessary guards and actions for the system transitions. The guards determine whether a process-loading or advancing event can take place, on the basis of the process and the resource availability. Upon the occurrence of such an event, the corresponding actions update accordingly the available resource units and the process instances that are active at the various processing stages. For a more detailed discussion on the EFA-based modeling of RAS and the above example, the reader is referred to Fei et al. (2011), Fei et al. (2013).

Some further remarks We remind the reader that every legitimate resource allocation state of the considered RAS must adhere to the restrictions that are imposed by the limited capacities of the system resources. In the representation of the EFA $E(\Phi)$, these restrictions are expressed by the following constraints

$$\forall i \in \{1, \ldots, m\}, \quad vR_i + \sum_{j=1}^{n} \sum_{k=1}^{l(j)-1} \mathcal{A}_{jk}[i] * v_{jk} = C_i \qquad (1)$$

In (1), we have taken into consideration the fact that terminal processing stages are not explicitly accounted for in the considered EFA model (for the reasons explained earlier). From a more technical standpoint, the constraints of (1) can be perceived as a set of (resource-induced) *invariants* that must be observed by the dynamics of the EFA $E(\Phi)$ in order to provide a faithful representation of the actual RAS dynamics. Hence, in the following, we shall characterize a state s of the EFA $E(\Phi)$ with a variable vector v satisfying the constraints of (1) as a *feasible* state.

On the other hand, the specification of the state set Q as $Q = L \times \bigotimes_i \mathcal{D}(vR_i) \times \bigotimes_{j,k} \mathcal{D}(v_{jk})$, where the domain sets $\mathcal{D}(v)$ of the various variables v are determined as described in the previous paragraphs, implies that Q may contain infeasible states, as well. The various guards and actions that are employed by EFA $E(\Phi)$ ensure that these infeasible states remain unreachable in any proper execution of $E(\Phi)$ that starts from some feasible state. More generally, as it will be revealed in the following, the infeasible states that might be included in EFA $E(\Phi)$ do not compromise the analytical power of this model regarding the traced RAS dynamics.

2.2 Binary Decision Diagrams

Binary decision diagrams (BDDs) (Bryant (1992)) are a memory-efficient data structure used to represent Boolean functions as well as to perform set-based operations. To present the basic BDD theory employed in this work, let us set $\mathbb{B} \equiv \{0, 1\}$. Also, for any Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}$, in n Boolean variables $X = (x_1, \ldots, x_n)$, we denote by $f|_{x_i=0(\text{resp. } 1)}$ the Boolean function that is induced from function f by fixing the value of variable x_i to 0 (resp. 1). Then, a BDD-based representation of f is a graphical representation of this function that is based on the following identity:

$$\forall x_i \in X,\; f = (\neg x_i \wedge f|_{x_i=0}) \vee (x_i \wedge f|_{x_i=1}) \qquad (2)$$

More specifically, (2) enables the representation of the Boolean function f as a single-rooted acyclic digraph with two types of nodes: *decision nodes* and *terminal nodes*. A terminal node can be labeled either 0 or 1. Each decision node is labelled by a Boolean variable and it has two outgoing edges, with each edge corresponding to assigning the value of the labeling variable to 0 or to 1. The value of function f for any given pricing of the variable set X is evaluated by starting from the root of the BDD and at each visited node following the edge that corresponds to the selected value for the node-labeling variable; the value of f is the value of the terminal node that is reached through the aforementioned path.

The *size* of a BDD refers to the number of its decision nodes. A carefully structured BDD can provide a more compact representation for a Boolean function f than the corresponding truth table and the decision tree; frequently, the attained compression is by orders of magnitude. From a computational standpoint, the power of BDDs lies in the efficiency that they provide in the execution of binary operations. Let f and f' be two Boolean functions of X. Then, it should be evident from (2) that a binary operator $\otimes$ between (the BDDs representing) f and f' can be recursively computed as

$$f \otimes f' = [\neg x \wedge (f|_{x=0} \otimes f'|_{x=0})] \vee [x \wedge (f|_{x=1} \otimes f'|_{x=1})] \quad (3)$$

where $x \in X$. If dynamic programming is used, the computation implied by (3) can have a complexity of $\mathcal{O}(|f| \cdot |f'|)$ where $|f|$ and $|f'|$ are the sizes of (the BDDs representing) f and f'.

A particular operator that is used extensively in the following is the *existential quantification* of a function f over its Boolean variables. For a variable $x \in X$, the existential quantification of f is defined by $\exists x.f = f|_{x=0} \vee f|_{x=1}$. Also, if $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_k) \subseteq X$, then $\exists \bar{X}.f$ is a shorthand notation for $\exists \bar{x}_1.\exists \bar{x}_2.\ldots\exists \bar{x}_k.f$. In plain terms, $\exists \bar{X}.f$ denotes all those truth assignments of the variable set $X \setminus \bar{X}$ that can be extended over the set $\bar{X}$ in a way that function f is eventually satisfied.

EFA encoding through BDDs To represent an EFA E by a Boolean function, different sets of Boolean variables are employed to encode the locations, events and integer variables. For the encoding of the state set $Q : L \times \mathcal{D}$, we employ two Boolean variable sets, denoted by X^L and $X^{\mathcal{D}} = X^{\mathcal{D}_1} \cup \ldots \cup X^{\mathcal{D}_n}$, to respectively encode the two sets L and $\mathcal{D}$. Then, each state $q = (\ell, v) \in Q$ is associated with a unique satisfying assignment of the variables in $X^L \cup X^{\mathcal{D}}$. Given a subset $\bar{Q}$ of Q, its *characteristic function*

$\chi_{\bar{Q}} : Q \to \{0,1\}$ assigns the value of 1 to all states $q \in \bar{Q}$ and the value of 0 to all states $q \notin \bar{Q}$.[3] The symbolic representation of the transition relation $\to$ relies on the same idea. A transition is essentially a tuple $\langle \ell, v, \sigma, \ell', v' \rangle$ specifying a source state $q = (\ell, v)$, an event σ, and a target state $q' = (\ell', v')$. Formally, we employ the variable sets X^L and $X^{\mathcal{D}}$ to encode the source state q, and a copy of X^L and $X^{\mathcal{D}}$, denoted by $\acute{X}^L$ and $\acute{X}^{\mathcal{D}}$ to encode the target state q'. In addition, we employ the Boolean variable set X^{Σ} to encode the alphabet of E, and we associate the event σ with a unique satisfying assignment of the variables in X^{Σ}. Then, we identify the transition relation $\to$ of E with the characteristic function

$$\Delta(\langle q, \sigma, q' \rangle) = \begin{cases} 1 \text{ if } \ell \xrightarrow{\sigma}_{g/a} \ell' \in \to, v \models g, v' = a(v) \\ 0 \text{ otherwise} \end{cases}$$

That is, Δ assigns the value of 1 to $\langle q, \sigma, q' \rangle$ if there exists a transition from ℓ to ℓ' labelled by σ, the values of the variables at ℓ satisfy the guard g, i.e., $v \models g$, and the values of the variables v' at ℓ' are the result of performing action a on v.

BDD-based modeling of the RAS behavior Given a RAS instance Φ and the distinct EFA $E_1, \ldots, E_n$ that model the resource allocation dynamics of the RAS process types $J_1, \ldots, J_n$, we shall denote by $\Delta_1, \ldots, \Delta_n$ the corresponding symbolic representations of $E(\Phi)$. The resource allocation dynamics generated by Φ can be formally expressed by the extended full synchronous composition (EFSC), introduced in Sköldstam et al. (2007), that composes the aforementioned EFA to the "plant" EFA $\mathbf{E} = E_1 || \ldots || E_n$. More specifically, in view of the above discussion on the infeasible states that might be contained in the EFA E_j, the actual dynamics of the considered RAS Φ are modeled by the subspace of the composed EFA $\mathbf{E}$ that is reachable from the (composed) initial state s_0. A symbolic representation of $\mathbf{E}$ will be denoted by $\Delta_{\mathbf{E}}$, and it can be perceived as a symbolic representation of the underlying RAS state-space (although containing all the "impurities" that were discussed earlier). $\Delta_{\mathbf{E}}$ can be systematically obtained from $\Delta_1, \ldots, \Delta_n$ by using the approach introduced in Miremadi et al. (2012); the discussion of this approach is beyond the scope of this work, and, thus, we refer to Miremadi et al. (2012) for the details.

Finally, as it will be revealed in the following, the computations pursued in this work do not require the explicit representation of the event set $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$. Hence, to reduce the number of Boolean variables employed by $\Delta_{\mathbf{E}}$, in the following we will suppress from $\Delta_{\mathbf{E}}$ the Boolean variable set $X^{\Sigma}_{\mathbf{E}}$, that represents Σ. In addition, since the locations of the considered EFA do not convey any substantial information other than characterizing the various process types as model entities with a distinct behavior modeled by the corresponding EFA, $\Delta_{\mathbf{E}}$ can be further compressed by suppressing the Boolean variable set $X^L_{\mathbf{E}} = X^L_1 \cup \ldots \cup X^L_n$, as well. The elimination of the aforementioned sets of variables from $\Delta_{\mathbf{E}}$ is technically effected through the following existential quantification:

$$\Delta_{\mathbf{E}} := \exists (X^{\Sigma}_{\mathbf{E}} \cup X^L_{\mathbf{E}}).\Delta_{\mathbf{E}} \qquad (4)$$

In the rest of this work, when we refer to the plant model $\Delta_{\mathbf{E}}$ we shall imply the output of the operation performed in (4).

[3] In the rest of this document, we shall use interchangeably the original name of a set Q and its characteristic function, χ_Q, in order to refer to this set.

3. THE MAIN ALGORITHMS

In this section, we present our symbolic approach for the retrieval of the set of minimal boundary unsafe states from the underlying RAS state-space. The presented algorithms assume the availability of an appropriately constructed BDD $\Delta_{\mathbf{E}}$ that is a valid representation of the composed EFA $\mathbf{E} = E_1 || \ldots || E_n$ and it has been compressed through the existential quantification expressed in (4). On the other hand, as it will be revealed in the sequel, the algorithms establish and maintain the feasibility of the various extracted states by utilizing the characteristic function χ_F that expresses state feasibility in the BDD-based representational context; this function can be systematically constructed by first expressing collectively the invariants of (1) through the Boolean function

$$\bigwedge_{i=1}^{m} \big(vR_i + \sum_{j=1}^{n} \sum_{k=1}^{l(j)-1} \mathcal{A}_{jk}[i] * v_{jk} = C_i\big), \qquad (5)$$

and subsequently setting χ_F equal to the BDD that collects the binary representations of all the value sets for the variables vR_i and v_{jk} that satisfy the Boolean function of (5).

The presented approach consists of two major stages. The first stage computes symbolically the entire set of the feasible boundary unsafe states of the given RAS, and, subsequently, the second stage extracts the minimal elements of this set. For reasons that are explained in the following, this particular subset is adequate for the representation of the entire original set in the implementation of the target DAP.

3.1 Computing the feasible boundary unsafe states

The symbolic algorithm for computing the feasible boundary unsafe states is also decomposed into two stages. In the first stage, all the deadlock states w.r.t. the advancement events in the considered RAS are identified and computed from the symbolic representation of the state space $\Delta_{\mathbf{E}}$. In the second stage, the deadlock states are used as starting points for a search procedure over $\Delta_{\mathbf{E}}$ that identifies all the boundary unsafe states. The entire computation is formally expressed by Algorithm 1, that works with $\Delta_{\mathbf{E}}$ and χ_F, and returns the characteristic functions χ_{FD} and χ_{FB} that constitute respective symbolic representations of the sets of the feasible deadlock states and the feasible boundary unsafe states. In general, the set χ_{FB} obtained from the presented algorithm may include some states that are not reachable from the initial state s_0. However, the presence of these additional states in the set χ_{FB} does not impede the implementation of the maximally permissive DAP by means of this set and the one-step-lookahead logic that was outlined in the earlier parts of this manuscript. Furthermore, for reasons that will become clear in the following, it is pertinent to assume that the characteristic function $\Delta_{\mathbf{E}}$ is partitioned in the characteristic functions Δ_A and Δ_L that collect respectively the transitions in $\Delta_{\mathbf{E}}$ corresponding to process advancement and process loading events; obviously, $\Delta_{\mathbf{E}} = \Delta_A \vee \Delta_L$. The rest of this section elaborates on the various phases of the computation that is depicted in Algorithm 1; a formal analysis of this algorithm, that establishes its finite termination and provides a proof for its correctness, can be found in Fei et al. (2013).

Identification of the feasible deadlock states. The symbolic operations for the computation of the characteristic function χ_{FD} are depicted in Lines 1-4 of Algorithm 1, and they can be described as follows: (i) The first step

Algorithm 1: Symbolic computation of the boundary unsafe states

```
Input: $\Delta_{\mathbf{E}}$ (as $\Delta_A \vee \Delta_L$) and $\chi_F$
Output: $\chi_{FB}$
$\chi_T := \big(\exists X^{\mathcal{D}}.\ (\Delta_A \vee \Delta_L)\big)[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}]$   1
$\chi_E := \exists \acute{X}^{\mathcal{D}}.\ \Delta_A$   2
$\chi_D := \chi_T \wedge \neg\chi_E \wedge \neg\chi_{\{s_0\}}$   3
$\chi_{FD} := \chi_D \wedge \chi_F$   4
$\chi_{U_{new}} := \chi_{FD}, \chi_U := \chi_{FD}, \Delta_{\hat{U}_{pre}} := 0$   5
repeat   6
    $\Delta_{\hat{U}} := \chi_{U_{new}}[X^{\mathcal{D}} \to \acute{X}^{\mathcal{D}}] \wedge \Delta_A$   7
    $\chi_{S\hat{U}} := \exists \acute{X}^{\mathcal{D}}.\ \Delta_{\hat{U}}$   8
    $\Delta_{SA} := \chi_{S\hat{U}} \wedge \Delta_A$   9
    $\chi_{NU} := \exists \acute{X}^{\mathcal{D}}.\ (\Delta_{SA} \wedge \neg\Delta_{\hat{U}} \wedge \neg\Delta_{\hat{U}_{pre}})$   10
    $\chi_{U_{cur}} := \chi_{S\hat{U}} \wedge \neg\chi_{NU}$   11
    $\chi_{U_{new}} := \chi_{U_{cur}} \wedge \neg\chi_U$   12
    $\chi_U := \chi_U \vee \chi_{U_{cur}}$   13
    $\Delta_{\hat{U}_{pre}} := (\Delta_{\hat{U}_{pre}} \vee \Delta_{\hat{U}}) \wedge \neg\chi_{U_{cur}}$   14
until $\chi_{U_{new}} = 0$   15
$\Delta_{\mathcal{B}} := \chi_U[X^{\mathcal{D}} \to \acute{X}^{\mathcal{D}}] \wedge \Delta_{\mathbf{E}}$   16
$\Delta_{S\mathcal{B}} := \Delta_{\mathcal{B}} \wedge (\neg\chi_U)$   17
$\chi_{FB} := (\exists X^{\mathcal{D}}.\ \Delta_{S\mathcal{B}})[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}]$   18
```

consists of Lines 1-3 and it computes the characteristic function χ_D of all the (partial) deadlock states in $\Delta_{\mathbf{E}}$, i.e., those states that are different from the initial state s_0 and they do not enable any process-advancing events. This function is computed by first extracting into the characteristic function χ_T all the target states from $\Delta_A \vee \Delta_L$ (i.e. from $\Delta_{\mathbf{E}}$) and in the characteristic function χ_E all the states that enable process-advancing events. Subsequently, χ_D is computed as the elements of χ_T that are not in χ_E (i.e., they do not enable any process-advancing events) or the initial state s_0.[4] (ii) Since χ_D is computed from the entire set of transitions that is contained in $\Delta_{\mathbf{E}}$, it might contain deadlock states that are infeasible (i.e., they violate the resource-induced invariants of (1)). The presence of these infeasible states in χ_D would increase unnecessarily the computational cost of the second stage of the considered algorithm, that utilizes the identified deadlock states as starting points for the identification of the additional set of deadlock-free unsafe states. Hence, in the last step of the first stage of Algorithm 1, the obtained state set χ_D is filtered through its conjunction with the characteristic function χ_F in order to obtain the set of feasible deadlock states; this set is represented by the characteristic function χ_{FD}.

Computation of the feasible boundary unsafe states. Having obtained the set χ_{FD} of the feasible deadlock states, the algorithm proceeds with the symbolic computation of the feasible boundary unsafe states in the RAS state-space $\Delta_{\mathbf{E}}$. These states are collected in the characteristic function χ_{FB}, which is computed in Lines 5-18 of Algorithm 1. A detailed description of this computation is as follows: (i) At this phase of the computation, Algorithm 1 employs the set U in order to collect all the identified unsafe states. Furthermore, at each iteration, the set U_{new} defines the set of the unsafe states that are to be processed at that iteration, through one-step-backtracking in Δ_A, in an effort to reach and explore new states. The corresponding symbolic representations for these two sets, denoted by χ_U and $\chi_{U_{new}}$, are initialized to χ_{FD}. Finally, we also define the transition set $\hat{U}_{pre} \equiv \{(s,u) \in \Delta_A \mid u \in U \wedge s \notin U\}$; i.e., during the entire search process, $\hat{U}_{pre}$ contains the transitions of Δ_A where the target states belong to U while the source states have also transitions to states that currently are not in U. The characteristic function of $\hat{U}_{pre}$ is initialized to zero. (ii) During the main iteration of the executed search process, the algorithm first extracts all the states that can be reached from the unsafe state set U_{new} by tracing backwards some process-advancing transition in Δ_A. This computation is performed in Lines 7-8 of the algorithm, with the extracted states represented by the characteristic function $\chi_{S\hat{U}}$. Also, the backtraced transitions of Δ_A are represented by the characteristic function $\Delta_{\hat{U}}$. (iii) Subsequently, Algorithm 1 tries to resolve which of the states collected in $\chi_{S\hat{U}}$ can be classified as unsafe. This resolution is performed in Lines 9-11 of the algorithm. More specifically, the algorithm first collects in the transition set Δ_{SA} all those process-advancing transitions of Δ_A that emanate from states in $\chi_{S\hat{U}}$. Subsequently, it removes from Δ_{SA} those transitions that are known to lead to unsafe states, namely the transitions that are also in $\Delta_{\hat{U}}$ and in $\hat{U}_{pre}$. The source states for any transitions remaining in Δ_{SA} after this last operation are collected in χ_{NU}; these are states that have transitions leading to states currently not in U, and therefore, they cannot be classified as unsafe (at least in this iteration). On the other hand, the complement of χ_{NU} w.r.t. the overall set of extracted states $\chi_{S\hat{U}}$ must contain states with all their emanating transitions leading to unsafe states, and therefore, they are themselves unsafe; these states are identified and collected in set $\chi_{U_{cur}}$ in Line 11. (iv) Lines 12-14 perform the necessary updates so that all the critical data structures represent correctly the current outcome of the ongoing search process. Hence, Line 12 removes from $\chi_{U_{cur}}$ any states that have already been classified as unsafe in the previous iterations; the remaining states are the elements of U_{new} for the next iteration. Line 13 adds to the set U the newly identified unsafe states, and finally, Line 14 updates the transition set $\hat{U}_{pre}$; this last update is performed by initially adding to $\hat{U}_{pre}$ all the transitions in $\Delta_{\hat{U}}$ (i.e., the transitions that were backtraced during the current iteration), and subsequently removing those transitions with source states identified as unsafe. (v) The iteration described in items (ii-iv) above terminates when no new unsafe states can be identified by the algorithm. At this point, Algorithm 1 proceeds to extract the boundary unsafe states from set χ_U. For that, at Line 16, the algorithm computes from $\Delta_{\mathbf{E}}$ all the transitions with the target states belonging to the unsafe state set χ_U; the relevant transition set is denoted by $\Delta_{\mathcal{B}}$. Next, at Line 17, the algorithm retrieves from $\Delta_{\mathcal{B}}$ the transition set $\Delta_{S\mathcal{B}}$, where the source states of the included transitions are safe states. Finally, χ_{FB} is obtained by extracting the target states from $\Delta_{S\mathcal{B}}$ and performing the replacement of $\acute{X}^{D}$ by $X^{\mathcal{D}}$.

3.2 Computing the minimal boundary unsafe states

An important implication of the invariants of (1) is that, at any feasible state of the RAS state-space, the values

[4] The particular operation $[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}]$, that is involved in the first step of the presented computation, moves the values that are obtained during that step from the variable set $\acute{X}^{\mathcal{D}}$ to the corresponding variables in the variable set $X^{\mathcal{D}}$, in order to be utilized in the subsequent steps of the algorithm; c.f. also Step 3.

of the resource variables vR_i can be induced from the values of the instance variables v_{jk}. Hence, one can obtain a more compact symbolic representation of the set of feasible boundary unsafe states, χ_{FB}, that is computed by Algorithm 1, by eliminating from the elements of χ_{FB} the values that correspond to the variables vR_i. Letting X^R denote the Boolean variables representing the values of the resource variables vR_i, $i = 1, \ldots, m$, this elimination can be performed through the following existential quantification:

$$\chi_{FB} := \exists X^R.\ \chi_{FB}. \tag{6}$$

The compressed representation of the set χ_{FB} that is obtained through (6) becomes even more important when noticing that, according to Reveliotis and Nazeem (2013), state unsafety is a monotone property in this representation. More specifically, given any two feasible boundary unsafe states u_1, u_2 represented according to the logic of (6), we consider the ordering relation "$\leq$" on them that is defined by the application of this relation componentwise; i.e.,

$$u_1 \leq u_2 \iff (\forall k = 1, \ldots, K, u_1[k] \leq u_2[k]), \tag{7}$$

where $u_1[k]$ and $u_2[k]$ are the values of the k-th instance variable for u_1 and u_2. Furthermore, we use the notation '$<$' to denote that condition (7) holds as strict inequality for at least one component $v_k \in \{v_1, \ldots, v_K\}$. It is shown in Reveliotis and Nazeem (2013) that if state u_1 is unsafe and state u_2 satisfies $u_1 \leq u_2$, then state u_2 is also unsafe. Hence, under the state representation of (6), the set FB can be effectively defined by the subset of its minimal elements, a realization that leads to ever greater economies regarding the storage and processing of the relevant information. We shall denote this subset by $\overline{FB}$, i.e., $\overline{FB} \equiv \{u \in FB \mid \nexists u' \in FB \text{ s.t. } u' < u\}$.

In the rest of this section we present an algorithm for computing the characteristic function of the set $\overline{FB}$ from the characteristic function χ_{FB} obtained in (6). Before we proceed with the discussion of this algorithm, we need to introduce two auxiliary BDD sets, collectively denoted by $\{\Delta_=(\tilde{v}_1, v_1), \ldots, \Delta_=(\tilde{v}_K, v_K)\}$ and $\{\Delta_\geq(\tilde{v}_1, v_1), \ldots, \Delta_\geq(\tilde{v}_K, v_K)\}$, which will be useful for identifying state dominances, according to (7), by the proposed algorithm. Each pair $\Delta_=(\tilde{v}_k, v_k)$ and $\Delta_\geq(\tilde{v}_k, v_k)$ pertains to the corresponding instance variable v_k, and it can be constructed as follows:

$$\Delta_=(\tilde{v}_k, v_k) := \bigvee_{\forall v_k \in \mathcal{D}_k} \left(\tilde{X}^{\mathcal{D}_k}(v_k) \wedge X^{\mathcal{D}_k}(v_k)\right) \tag{8}$$

$$\Delta_\geq(\tilde{v}_k, v_k) := \bigvee_{\forall v_k \in \mathcal{D}_k} \left(\tilde{X}^{\mathcal{D}_k}(v_k) \wedge \bigvee_{\forall v'_k \geq v_k} X^{\mathcal{D}_k}(v'_k)\right) \tag{9}$$

In (8) and (9), $\tilde{X}^{\mathcal{D}_k}(v_k)$ denotes the symbolic representation of the value of k-th variable v_k using a new set of Boolean variables denoted by $\tilde{X}^{\mathcal{D}_k}$ while $X^{\mathcal{D}_k}(v_k)$ and $X^{\mathcal{D}_k}(v'_k)$ denote the symbolic representations of the values v_k and v'_k, of the same instance variable, using the source Boolean variables $X^{\mathcal{D}_k}$. From a conceptual standpoint, $\Delta_\geq(\tilde{v}_k, v_k)$ associates each value v_k with all those values $v'_k \in \mathcal{D}_k$ that are greater than or equal to v_k while $\Delta_=(\tilde{v}_k, v_k)$ merely associates each value v_k with itself.

Taking as input the feasible boundary unsafe state set χ_{FB} and the aforementioned auxiliary BDDs, the symbolic computation of the minimal feasible boundary unsafe states is formally expressed by Algorithm 2. Specifically, in Lines 1-2, Algorithm 2 constructs two BDDs, respectively denoted by Δ_{EQ} and Δ_{GE}, by performing the conjunction operation on $\{\Delta_\geq(\tilde{v}_1, v_1), \ldots, \Delta_\geq(\tilde{v}_K, v_K)\}$ and $\{\Delta_=(\tilde{v}_1, v_1), \ldots, \Delta_=(\tilde{v}_K, v_K)\}$. The characteristic function Δ_{EQ} associates each state $\langle v_1, \ldots, v_K\rangle$ with two different symbolic representations using the Boolean variable sets $\tilde{X}^{\mathcal{D}}$ and $X^{\mathcal{D}}$, while Δ_{GE} associates each state $\langle v_1, \ldots, v_K\rangle$, represented by $\tilde{X}^{\mathcal{D}}$, with a set of states, represented by $X^{\mathcal{D}}$, which are larger than or equal to $\langle v_1, \ldots, v_K\rangle$. Subsequently, the symbolic computation performed at Line 3 of Algorithm 2 removes all the associations of Δ_{EQ} from Δ_{GE} and denotes by Δ_{GT} the resulting set. Line 4 of Algorithm 2 computes the characteristic function Δ_{BGT} which associates each state in χ_{FB} with the corresponding dominant states, and, subsequently, Line 5 extracts all these dominant states into the set χ_{GB}. Finally, the set of minimal feasible boundary unsafe states, $\chi_{\overline{FB}}$, is obtained in Line 6 by removing from χ_{FB} the states in χ_{GB}. A formal proof for the correctness of Algorithm 2 can be found in Fei et al. (2013).

Algorithm 2: Symbolic computation of the minimal boundary unsafe states

```
Input: $\chi_{FB}$, $\{\Delta_=(\tilde{v}_1, v_1), \ldots, \Delta_=(\tilde{v}_K, v_K)\}$ and
       $\{\Delta_\geq(\tilde{v}_1, v_1), \ldots, \Delta_\geq(\tilde{v}_K, v_K)\}$
Output: $\chi_{\overline{FB}}$
$\Delta_{EQ} := \Delta_=(\tilde{v}_1, v_1) \wedge \ldots \wedge \Delta_=(\tilde{v}_K, v_K)$        1
$\Delta_{GE} := \Delta_\geq(\tilde{v}_1, v_1) \wedge \ldots \wedge \Delta_\geq(\tilde{v}_K, v_K)$        2
$\Delta_{GT} := \Delta_{GE} \wedge \neg\Delta_{EQ}$        3
$\Delta_{BGT} := \chi_{FB}[X^{\mathcal{D}} \to \tilde{X}^{\mathcal{D}}] \wedge \Delta_{GT}$        4
$\chi_{GB} := \exists\tilde{X}^{\mathcal{D}}.\ \Delta_{BGT}$        5
$\chi_{\overline{FB}} := \chi_{FB} \wedge \neg\chi_{BG}$        6
```

4. EXPERIMENTAL RESULTS

The symbolic algorithms that were developed in this work have been implemented in the DES software tool Supremica (Åkesson et al. (2006)). In this section, we report the results from a series of computational experiments [5] in which we applied the considered algorithms on a number of randomly generated instantiations of the RAS class that was defined in Section 1. Moreover, we compare the performance of Algorithm 1 to the performance of the BDD-based algorithm of Miremadi et al. (2012); the latter computes the target unsafe states through the standard synthesis procedure for non-blocking SC provided by SC theory (Cassandras and Lafortune (2008)).

Table 1 reports a representative sample of the results obtained in our experiments. The first section in the table is for RAS instances with simple linear process flows and single-type resource allocation. The second section is for RAS instances with simple linear process flows and conjunctive resource allocation. Finally, the last section of the table is for RAS instances with routing flexibility and conjunctive resource allocation. In the experiments, we first applied an extension of the algorithm presented in Miremadi et al. (2012) and Algorithm 1 to the aforementioned RAS instances to obtain all the boundary unsafe states, and then we applied Algorithm 2 to remove the non-minimal unsafe states from the obtained sets of boundary unsafe states. Columns 1-2 in Table 1 report the cardinalities of the set of reachable states R and the set of reachable

[5] The experiments were carried out on a standard desktop, (2.66 GHz Intel Core Quad CPU, 10GB RAM) running Windows 7

Table 1. A sample of computational results regarding the efficiency of the presented algorithms.

The algorithm of Miremadi et al. (2012)							Algorithm 1 presented in Section 3.1						
$\|R\|$	$\|RB\|$	t_{RB}	ζ_{RB}	$\|\overline{RB}\|$	$t_{\overline{RB}}$	$\zeta_{\overline{RB}}$	$\|FD\|$	$\|FB\|$	t_{FB}	ζ_{FB}	$\|\overline{FB}\|$	$t_{\overline{FB}}$	$\zeta_{\overline{FB}}$
635572	180603	5	220504	1513	0	68983	220976	341959	10	216780	2408	0	74850
1463878	227817	11	176658	284	0	22158	55001	324949	31	229082	509	0	32624
404542	95971	7	278983	3084	0	218063	281285	206867	8	226044	5319	0	189842
1508301	267871	2	247264	664	0	29361	133231	340325	3	216089	1011	0	26037
799071	186500	21	527040	6193	1	586529	291592	283962	7	283962	8934	1	551210
1743534	351907	15	479966	2366	0	250208	651211	615057	18	355962	3653	0	361054
1659342	381846	90	1115932	9472	19	3943512	942254	800940	42	796123	17931	7	2046115
1962454	438521	28	607812	6719	2	853680	769090	761399	29	450040	10527	4	929955
4488904	860221	15	499894	642	0	53755	1032734	1560858	75	989925	1928	0	76711
3436211	783794	207	1374268	31236	41	7775958	1590736	1564991	106	1176110	55553	73	8985355
14158338	2615904	180	1731691	31629	11	2709199	1983934	3558362	152	1561971	46048	18	2939342
14521572	3218012	626	3556004	26920	34	8969555	3399416	5696085	642	4999572	51069	102	7920050
22212582	5066271	2150	8019401	31328	60	10435459	4621662	8056766	964	5546176	62996	238	12958398
32380375	8277582	609	4347910	21062	20	2719450	4807088	14320225	904	5415820	40306	109	4506280
14963458	3207511	470	3207511	17990	104	12698346	6898234	5989367	553	4415000	31376	393	17578012
29160898	5496694	184	2126861	15957	11	1976633	3035820	7751451	237	2685162	27138	25	2442037
646746	94821	0	65495	779	0	12588	23876	134569	0	39073	849	0	14890
1767552	308306	0	137372	559	0	4418	29616	351851	0	90427	660	0	4979
915716	166540	2	219628	794	0	54459	199316	276548	11	325337	1072	0	55987
738720	69536	3	260919	4091	0	39553	171581	82408	3	166049	4616	0	60080
2939463	408009	128	1432070	3401	0	97229	142301	531238	97	1043925	5464	0	80869
2430581	547612	15	505697	14120	0	189566	247195	741764	10	226991	16732	0	104188
1962454	438521	33	613613	6719	1	678536	769080	761399	25	649984	10527	2	996436
1712672	306585	83	1590899	6821	0	175493	441376	445092	38	646998	9563	3	318205
3554952	614597	3	282374	2466	0	43964	642882	1212213	34	467604	3520	0	68264
6051299	1087093	134	1547620	4713	0	139537	1189993	1781191	32	575720	6292	0	157214
24430444	5457497	205	2408072	9491	0	323161	1037721	6000747	125	1534599	10699	0	228345
2271288	532934	18	458589	2636	0	261719	771400	947149	21	629325	5601	0	395819
29160898	5496694	211	1898949	15957	11	1976636	3035820	7751451	193	2146384	27138	12	1535306
22212582	5066271	946	6665791	31328	61	10472017	4621662	8056766	815	5182290	62996	244	13028200
106509798	10910823	234	3873700	4035	0	69722	841940	12529669	313	2367893	4368	0	43553
596212152	139238562	3097	6791929	426	0	86650	2033997	169402134	520	6744437	572	0	50278
571536	0	0	189732	0	0	0	0	0	0	4908	0	0	0
2771880	11925	0	367188	28	0	1715	1445	12306	0	21102	28	0	1806
1229688	26088	13	663513	241	0	2019	21046	39120	0	112433	271	0	2558
2693250	29286	1	108440	79	0	1363	3193	29292	0	64316	79	0	1433
3416000	78400	0	269513	1	0	288	49	78400	0	80758	1	0	315
2448000	330768	1	129663	9	0	1872	10076	336144	2	255463	9	0	1881
1663534	130825	1	175736	2665	0	27102	185177	262514	1	129084	6189	0	30140
2340408	342098	12	525047	1458	0	15405	114926	603701	2	230807	2283	0	13289
7885856	425741	28	1199596	2323	0	17084	383129	594828	1	262861	2628	0	11999
30397584	568889	24	3544487	16526	0	62048	349953	853537	3	229892	22318	0	39358
1219947240	18531807	4987	49835897	381	0	639461	86535	72055380	460	7959586	516	0	495417
81285120	2027904	2	314728	1215	0	1971	110656	4676480	0	120387	1245	0	1860
96438720	5401790	365	3031243	24	0	11703	1648506	6321838	106	2526813	31	0	6592
3547065654	41135520	3892	20605813	6635	0	57369	1812728	93980859	74	3595817	8117	0	92971
399477600	45541152	2802	14511314	792	0	58762	2027551	122636544	59	2939165	1975	0	20899
3749923584	222163176	8773	38249085	2320	0	343680	4177807	269219724	99	2441987	5171	0	189467

boundary unsafe states RB that are computed by the algorithm of Miremadi et al. (2012). Columns 3-4 report the required computation time, t_{RB} (in seconds), and the maximal number of BDD nodes, ζ_{RB}, employed during the algorithm execution. Having the set of all reachable boundary unsafe states, Column 5 reports the cardinality of the set of the minimal boundary unsafe states $\overline{RB}$, obtained through Algorithm 2. Columns 6-7 show the computation time $t_{\overline{RB}}$ and the maximal number of BDD nodes, $\zeta_{\overline{RB}}$, during the execution of Algorithm 2. With respect to the related results of Algorithm 1, Columns 8-9 report the cardinalities of the set of deadlock states FD and the set of boundary unsafe states FB, while Columns 10-11 report the computation time t_{FB} and the peak of the BDD nodes during the execution of Algorithm 1. Analogously, Columns 12-14 report the cardinality of the set of minimal boundary unsafe states $\overline{FB}$, the computation time $t_{\overline{FB}}$, and the maximal number of BDD nodes $\zeta_{\overline{FB}}$ when using Algorithm 2 to remove the non-minimal states from FB.

Thanks to the compactness offered by the employed symbolic representations, both, the symbolic algorithm of Miremadi et al. (2012) and Algorithm 1 are capable of handling RAS instances that have billions of states in their underlying state-spaces, and they manage to compute the set of boundary unsafe states with limited memory and time. Table 1 also reveals that functions like the removal of non-minimal boundary unsafe states from the originally computed sets RB and FB can be performed very efficiently through symbolic computation.

Next, we focus on the comparison of the computation time and the maximal memory usage between the algorithm of Miremadi et al. (2012) and Algorithm 1. By taking advantage of the particular structure and properties of the considered RAS state spaces, Algorithm 1 avoids the full exploration of these state-spaces. Hence, compared to the more conventional symbolic algorithm of Miremadi et al. (2012), Algorithm 1 requires fewer iterations to compute the target boundary unsafe states, and it tends to have a better computation time. Furthermore, the avoidance of the exploration of the whole RAS state-space enables Algorithm 1 to consume less memory during its execution, especially for RAS instances with small unsafe state regions. As depicted in Figures 3 and 4, Algorithm 1 outperforms the algorithm of Miremadi et al. (2012), on average. This performance dominance is more emphatic for RAS instances with routing flexibility, since, for these RAS, the cardinality of the set of reachable states is orders of magnitude larger than that of the set of boundary unsafe states, and therefore, the partial state-space exploration that is effected by Algorithm 1 establishes a stronger competitive advantage.

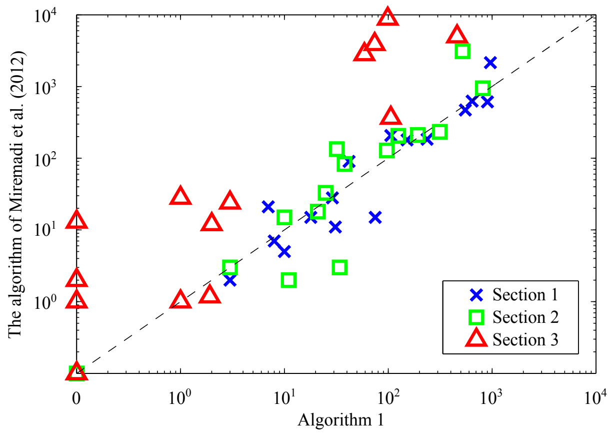

Fig. 3. Comparing the computation times (in sec.) of the algorithm of Miremadi et al. (2012) and Algorithm 1

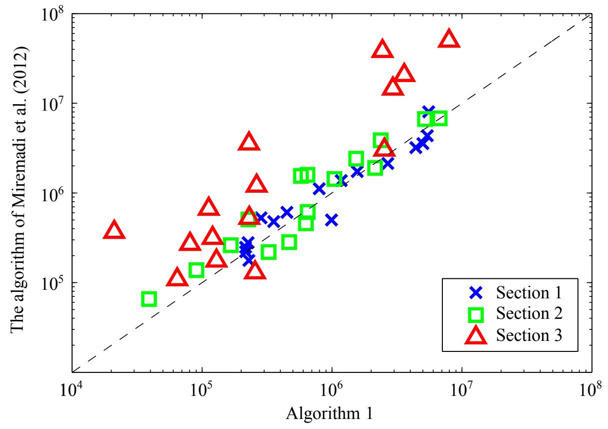

Fig. 4. Comparing the maximal memory usage of the algorithm of Miremadi et al. (2012) and Algorithm 1, based on their maximal BDD node requirements

Finally, we briefly report on some further experimental results presented in Fei et al. (2013), that compare the total computation time of the combined execution of Algorithms 1 and 2 to the computation time of the algorithm that is presented in (Nazeem and Reveliotis (2014)). According to the findings reported in Fei et al. (2013), for the RAS instances with simple linear process flows and conjunctive resource allocation, the sequential execution of Algorithms 1 and 2 outperforms the algorithm of Nazeem and Reveliotis (2014). Some of the largest cases suggest that the gains attained by the symbolic algorithms can be up to 100 times faster. On the other hand, for RAS instances possessing routing flexibility, the algorithm of Nazeem and Reveliotis (2014) is competitive to the proposed algorithms. We believe that this comparative improvement of the computational efficiency of the algorithm of Nazeem and Reveliotis (2014) for these particular RAS instances stems from the fact that the algorithm of Nazeem and Reveliotis (2014) focuses explicitly upon *minimal* deadlocks and unsafe states in its computation, and therefore, it effects an even more limited search in the underlying RAS state space compared to the algorithms that are developed herein.

5. CONCLUSION

This paper has complemented recent developments concerning the effective deployment of the maximally permissive DAP for complex RAS, with the representational and computational strengths that are offered by symbolic computation based on BDDs. The reported experimental results also demonstrate the substantial gains that can be attained, even in the context of symbolic computation, through the pertinent exploitation of the special structure that might be inherent in the underlying problem.

In our future work, we shall seek to further enhance the performance of the presented algorithms by identifying and exploiting additional special structure that might be present in the considered supervisory control problem, and we shall also extend these algorithms so that they can be applied to RAS with even more complex behavior.

REFERENCES

Åkesson, K., Fabian, M., Flordal, H., and Malik, R. (2006). Supremica - An integrated environment for verification, synthesis and simulation of discrete event systems. In *the 8th International Workshop on Discrete Event Systems*, 384–385.

Bryant, R.E. (1992). Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3), 293–318.

Campos, J., Seatzu, C., and Xie, X. (eds.) (2014). *Formal Methods in Manufacturing.* CRC Press.

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems.* Springer, 2nd edition.

Fei, Z., Reveliotis, S., Miremadi, S., and Åkesson, K. (2013). A BDD-based approach for designing maximally permissive deadlock avoidance policies for complex resource allocation systems. Technical report, Chalmers University. `http://publications.lib.chalmers.se/records/fulltext/186774/local_186774.pdf`.

Fei, Z., Miremadi, S., and Åkesson, K. (2011). Modeling sequential resource allocation systems using extended finite automata. In *7th Annual IEEE Conference on Automation Science and Engineering, CASE'11*, 444–449. Trieste.

Miremadi, S., Lennartson, B., and Åkesson, K. (2012). A BDD-based approach for modeling plant and supervisor by extended finite automata. *IEEE Transactions on Control Systems Technology*, 20(6), 1421–1435.

Nazeem, A. and Reveliotis, S. (2014). Efficient enumeration of minimal unsafe states in complex resource allocation systems. *IEEE Trans. on Automation Science and Engineering*, 11(1), 111–124.

Ramadge, P. and Wonham, W.M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1), 81–98.

Reveliotis, S. and Nazeem, A. (2013). Deadlock avoidance policies for automated manufacturing systems using finite state automata. In J. Campos, C. Seatzu, and X. Xie (eds.), *Formal Methods in Manufacturing*, 169–195. CRC Press / Taylor and Francis.

Reveliotis, S.A. (2005). *Real-time Management of Resource Allocation Systems: A Discrete Event Systems Approach.* Springer, NY, NY.

Sköldstam, M., Åkesson, K., and Fabian, M. (2007). Modeling of discrete event systems using finite automata with variables. *Decision and Control, 2007 46th IEEE Conference on*, 3387–3392.

Zhou, M. and Fanti, M.P. (2004). *Deadlock Resolution in Computer-Integrated Systems.* Marcel Dekker, Inc., Singapore.